Find the Pivot in Rotated Sorted array

IV semester - Bachelor's of Technology in Information technology with specialization in Business Informatics,

Indian Institute of Information Technology Allahabad, India

1st Amanjeet Kumar
IIB2019006
iiB2019006@iiita.ac.in
Amanjeetk11

2nd Aditya Raj
IIB2019007
iib2019007@iiita.ac.in
Adityahulk

3rd Shyam Tayal
IIB2019008
iib2019008@iiita.ac.in
shyamTayal

***Abstract*—In this paper, we are devising an algorithm to find the Rotation Count in Rotated (clockwise) Sorted array. This paper also analyzes the time and space complexity of the algorithms used and provides the most efficient approach to solve the given problem.**

***Index Terms*—array, sorted, rotated, clockwise, time complexity, space complexity, Divide and Conquer**

I. Introduction

An array is a collection of items stored at contiguous memory locations. The binary search algorithm uses Divide And Conquer approach which can be divided in three parts - Divide: This involves dividing the problem into some sub problem. Conquer: Sub problem by calling recursively until sub problem solved. Combine: The Sub problem Solved so that we will get find problem solution.

II. Algorithm Design

We have devised two algorithms to Find the Rotation Count in Rotated Sorted array.

The First algorithm uses linear search to find the index of minimum element. As we can notice that the number of rotations is equal to index of minimum element. It is a simple approach is to find minimum element and returns its index.

Algorithm 1: Naive Algorithm (Linear Search)

```
function findPivot(a[], size):
      j = 0
      loop i in range (0 to size-1):
          if i > 0 and a[i-1] > a[i]:
              return j
```

The second algorithm uses divide and conquer approach. Here also we find the index of minimum element, but using Binary Search.

The minimum element is the only element whose previous is greater than it.

If there is no previous element, then there is no rotation (first element is minimum).

We check this condition for middle element by comparing it with (mid)'th and (mid+1)'th elements.

If the minimum element is not at the middle (neither mid nor mid + 1), then minimum element lies in either left half or right half.

If middle element is smaller than last element, then the minimum element lies in left half

Else minimum element lies in right half.

Algorithm 2: Eficient Algorithm(Divide and Conquer)

```
function findPivot(arr[], N):
    if arr[0] <= arr[N-1] :
        return 0

    start = 0 , end = N-1;
    while start <= end  :
        mid = start + (end - start)/2

        if arr[mid] > arr[mid+1] :
            return mid+1

        if arr[start] <= arr[mid] :
            start = mid + 1
        else
            end = mid - 1

    return start
```

III. Algorithm Analysis

A. Time Complexity

1) Algorithm 1:
Best Case: $\Omega(n)$
Average Case: $\Theta(n)$
Worst Case: O(n)

2) Algorithm 2:
Best Case: $\Omega(1)$
Average Case: $\Theta(log(n))$
Worst Case: O($log(n)$)

B. Space Complexity

1) Algorithm 1:
O(1)

2) Algorithm 2:
O(1)

Random number generator has been used for filling the arrays in every case. Following graphs compare the times taken by the Linear Search as well as Divide and Conquer approaches.

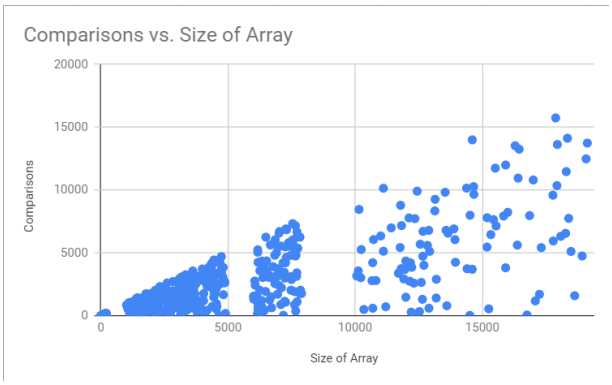

Fig. 1: Algorithm 1 VS N

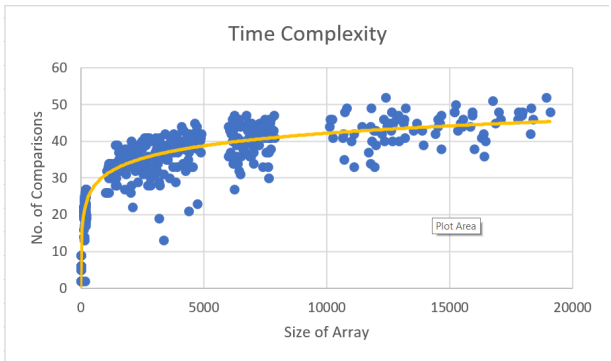

Fig. 2: Algorithm 2 VS N Linear Scale

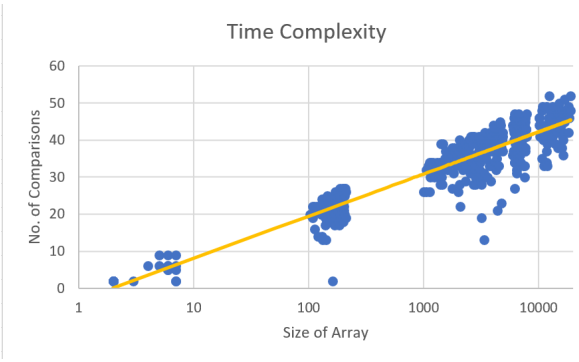

Fig. 3: Algorithm 2 VS N logarithmic Scale

IV. Conclusion

The more efficient algorithm, as we can see, turns out to be the 2nd algorithm (using binary search i.e. divide and conquer) against the 1st algorithm (linear search) as it has better complexity. The rotation count(pivot) will be minimum (i.e. 0) when array is already sorted.

References

[1] https://en.wikipedia.org/wiki/Binary_search_algorithm
[2] https://www.geeksforgeeks.org/divide-and-conquer-algorithm-introduction/